Introduction to the Command Line

Jan Moren, SCDA

OKINAWA INSTITUTE OF SCIENCE AND TECHNOLOGY GRADUATE UNIVERSITY

沖縄科学技術大学院大学

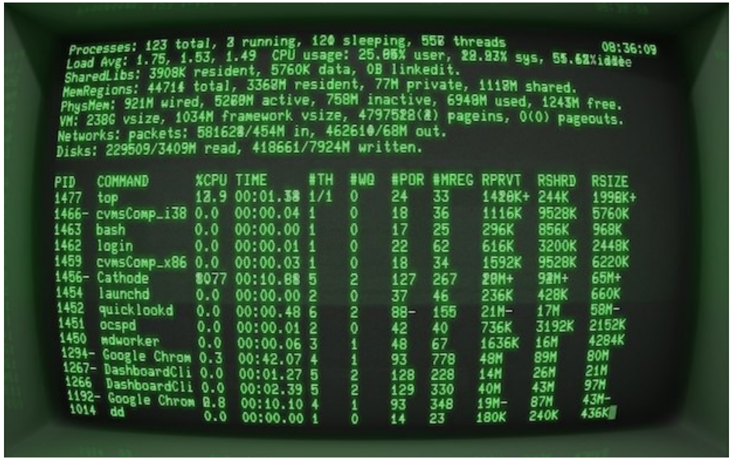
Set Up SSH

- Any OIST member can use the HPC resources. Apply here:
  - https://groups.oist.jp/scs/request-access
    Select “Open Resources”
- OSX Users
  - You should already have SSH
  - Install “XQuartz” for graphics (reboot after installation)
- Windows Users
  - Install free “MobaXTerm”. Can use SSH and graphical applications.
- Linux and BSD Users
  - You already have everything.

https://groups.oist.jp/scs/connect-clusters

The Command Line

We have a perfectly fine graphical environment.
Why do we bother with a command line?!



```
Processes: 123 total, 3 running, 120 sleeping, 558 threads                08:36:09
Load Avg: 1.75, 1.53, 1.49  CPU usage: 25.05% user, 20.93% sys, 55.62%idle
SharedLibs: 3908K resident, 5760K data, 0B linkedit.
MemRegions: 44714 total, 3360M resident, 77M private, 1110M shared.
PhysMem: 921M wired, 5200M active, 758M inactive, 6940M used, 1243M free.
VM: 238G vsize, 1034M framework vsize, 4797528(2) pageins, 0(0) pageouts.
Networks: packets: 581628/454M in, 462610/68M out.
Disks: 229509/3409M read, 418661/7924M written.

PID   COMMAND      %CPU TIME     #TH  #WQ  #POR #MREG RPRVT  RSHRD RSIZE
1477  top          12.9 00:01.38 1/1  0    24   33    1420K+ 244K  1990K+
1466- cvmsComp_i38 0.0  00:00.04 1    0    18   36    1116K  9528K 5760K
1463  bash         0.0  00:00.00 1    0    17   25    296K   856K  968K
1462  login        0.0  00:00.01 1    0    22   62    616K   3200K 2448K
1459  cvmsComp_x86 0.0  00:00.03 1    0    18   34    1592K  9528K 6220K
1456- Cathode      8077 00:10.88 5    2    127  267   20M+   92M+  65M+
1454  launchd      0.0  00:00.00 2    0    37   46    236K   428K  660K
1452  quicklookd   0.0  00:00.48 6    2    88-  155   21M-   17M   58M-
1451  ocspd        0.0  00:00.01 2    0    42   40    736K   3192K 2152K
1450  mdworker     0.0  00:00.06 3    1    48   67    1636K  16M   4284K
1294- Google Chrom 0.3  00:42.07 4    1    93   778   48M    89M   80M
1267- DashboardCli 0.0  00:01.27 5    2    128  228   14M    26M   21M
1266  DashboardCli 0.0  00:02.39 5    2    129  330   40M    43M   97M
1192- Google Chrom 0.8  00:10.10 4    1    93   348   19M-   87M   43M-
1014  dd           0.0  00:00.00 1    0    14   23    180K   240K  436K
```

Why The Command Line?

```
Processes: 123 total, 3 running, 120 sleeping, 550 threads                08:36:09
Load Avg: 1.75, 1.53, 1.49  CPU usage: 25.05% user, 20.03% sys, 55.62% idle
SharedLibs: 3908K resident, 5760K data, 0B linkedit.
MemRegions: 44714 total, 3360M resident, 77M private, 1110M shared.
PhysMem: 921M wired, 5200M active, 758M inactive, 6940M used, 1243M free.
VM: 238G vsize, 1034M framework vsize, 4797528(2) pageins, 0(0) pageouts.
Networks: packets: 581628/454M in, 462610/68M out.
Disks: 229509/3409M read, 418661/7924M written.

PID   COMMAND      %CPU TIME     #TH  #WQ  #POR #MREG RPRVT  RSHRD  RSIZE
1477  top          12.9 00:01.38 1/1  0    24   33    1420K+ 244K   1990K+
1466- cvmsComp_i38 0.0  00:00.04 1    0    18   36    1116K  9528K  5760K
1463  bash         0.0  00:00.00 1    0    17   25    296K   856K   968K
1462  login        0.0  00:00.01 1    0    22   62    616K   3200K  2448K
1459  cvmsComp_x86 0.0  00:00.03 1    0    18   34    1592K  9528K  6220K
1456- Cathode      8077 00:10.88 5    2    127  267   20M+   92M+   65M+
1454  launchd      0.0  00:00.00 2    0    37   46    236K   428K   660K
1452  quicklookd   0.0  00:00.48 6    2    88-  155   21M-   17M    58M-
1451  ocspd        0.0  00:00.01 2    0    42   40    736K   3192K  2152K
1450  mdworker     0.0  00:00.06 3    1    48   67    1636K  16M    4284K
1294- Google Chrom 0.3  00:42.07 4    1    93   778   48M    89M    80M
1267- DashboardCli 0.0  00:01.27 5    2    128  228   14M    26M    21M
1266  DashboardCli 0.0  00:02.39 5    2    129  330   40M    43M    97M
1192- Google Chrom 0.8  00:10.10 4    1    93   348   19M-   87M    43M-
1014  dd           0.0  00:00.00 1    0    14   23    180K   240K   436K
```

- It's Precise and Composable

  Combine commands to quickly do very complex tasks

- It's Scriptable

  Automate recurring tasks.

- It's Low Bandwidth

  Access from anywhere, work on any device.

Why **not** The Command Line?

```
Processes: 123 total, 3 running, 120 sleeping, 550 threads              08:36:09
Load Avg: 1.75, 1.53, 1.49  CPU usage: 25.05% user, 20.93% sys, 55.62%idle
SharedLibs: 3908K resident, 5760K data, 0B linkedit.
MemRegions: 44714 total, 3360M resident, 77M private, 1110M shared.
PhysMem: 921M wired, 5200M active, 758M inactive, 6940M used, 1243M free.
VM: 238G vsize, 1034M framework vsize, 4797528(2) pageins, 0(0) pageouts.
Networks: packets: 581620/454M in, 462610/68M out.
Disks: 229509/3409M read, 418661/7924M written.

PID   COMMAND      %CPU TIME     #TH  #WQ  #POR #MREG RPRVT  RSHRD  RSIZE
1477  top          12.9 00:01.38 1/1  0    24   33    1420K+ 244K   1998K+
1466- cvmsComp_i38 0.0  00:00.04 1    0    18   36    1116K  9528K  5760K
1463  bash         0.0  00:00.00 1    0    17   25    296K   856K   968K
1462  login        0.0  00:00.01 1    0    22   62    616K   3200K  2448K
1459  cvmsComp_x86 0.0  00:00.03 1    0    18   34    1592K  9528K  6220K
1456- Cathode      8077 00:10.88 5    2    127  267   20M+   92M+   65M+
1454  launchd      0.0  00:00.00 2    0    37   46    236K   428K   660K
1452  quicklookd   0.0  00:00.48 6    2    88-  155   21M-   17M    58M-
1451  ocspd        0.0  00:00.01 2    0    42   40    736K   3192K  2152K
1450  mdworker     0.0  00:00.06 3    1    48   67    1636K  16M    4284K
1294- Google Chrom 0.3  00:42.07 4    1    93   778   48M    89M    80M
1267- DashboardCli 0.0  00:01.27 5    2    128  228   14M    26M    21M
1266  DashboardCli 0.0  00:02.39 5    2    129  330   40M    43M    97M
1192- Google Chrom 0.8  00:10.10 4    1    93   348   19M-   87M    43M-
1014  dd           0.0  00:00.00 1    0    14   23    180K   240K   436K
```

- It's Opaque and hard to Explore

  What is even available?! What can you do?!

- It's Intimidating

  Feels like you can break things at any time.

- It's (sometimes) not Accessible

  You need a keyboard and you need to be able to use it.

Command Line

We show our command line examples like this:

```
# log in to Deigo
$ ssh -X your-name@deigo.oist.jp
```

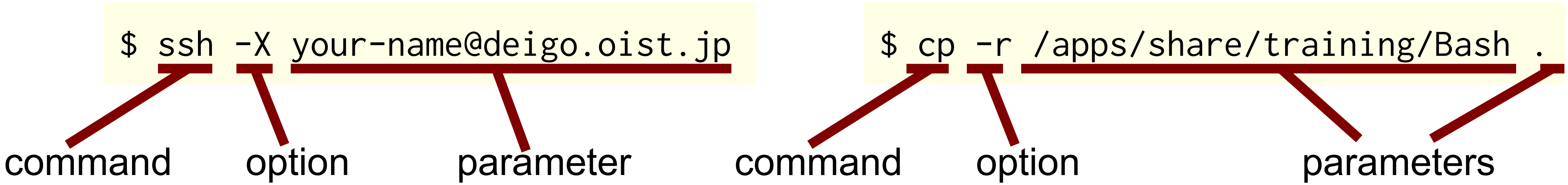
- "$" is a prompt where you can type things. We use it to show commands you run.
- "#" starts a Bash comment.

Command Line Examples

```
# log in to Deigo
$ ssh -X your-name@deigo.oist.jp

# copy training data to your home
$ cp -r /apps/share/training/Bash .

# Edit a slurm file with the 'nano' editor
$ nano my-script.slurm

# run the firefox browser
$ firefox
```

Components:

- command
- options
- parameters



```
$ ssh -X your-name@deigo.oist.jp
```

```
$ cp -r /apps/share/training/Bash .
```

Command Line Examples

```
# log in to Deigo
$ ssh -X your-name@deigo.oist.jp

# copy training data to your home
$ cp -r /apps/share/training/Bash .

# Edit a slurm file with the 'nano' editor
$ nano my-script.slurm

# run the firefox browser
$ firefox
```

Components:

- **command**
- options
- parameters

Command: The name of an application

- Most are programs
- A few are built in to the shell itself

Command Line Examples

```
# log in to Deigo
$ ssh -X your-name@deigo.oist.jp

# copy training data to your home
$ cp -r /apps/share/training/Bash .

# Edit a slurm file with the ’nano’ editor
$ nano my-script.slurm

# run the firefox browser
$ firefox
```

Components:

- command
- **options**
- parameters

Options: Changes how the program works

- One "-" and one letter ("`-X`"), or two "--" and a word ("`--verbose`")
- Short options can often be combined: "`-s -t`" → "`-st`"
- Options can have values ("`--user jan-moren`")

Command Line Examples

```
# log in to Deigo
$ ssh -X your-name@deigo.oist.jp

# copy training data to your home
$ cp -r /apps/share/training/Bash .

# Edit a slurm file with the 'nano' editor
$ nano my-script.slurm

# run the firefox browser
$ firefox
```

Components:

- command
- options
- **parameters**

Parameters: What the application should work on

- ssh: name of remote computer
- copy: source and destination
- text editor: name of file to edit

Let’s Log In

Log in to Deigo:

```
  $ ssh -X your-name@deigo.oist.jp
```

copy the slides, example scripts and programs to your home:

```
  $ cp -r /apps/share/training/Bash .
```

Let’s Log In

Log in to Deigo:

```
  $ ssh -X your-name@deigo.oist.jp
```

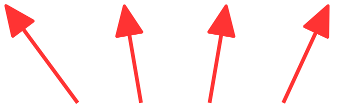
copy the slides, example scripts and programs to your home:

```
  $ cp -r /apps/share/training/Bash .
```

Handy Tip:

Avoid typing with *tab completion:*

```
$ cp -r /a<tab>/sh<tab>/t<tab>/B<tab> .
```

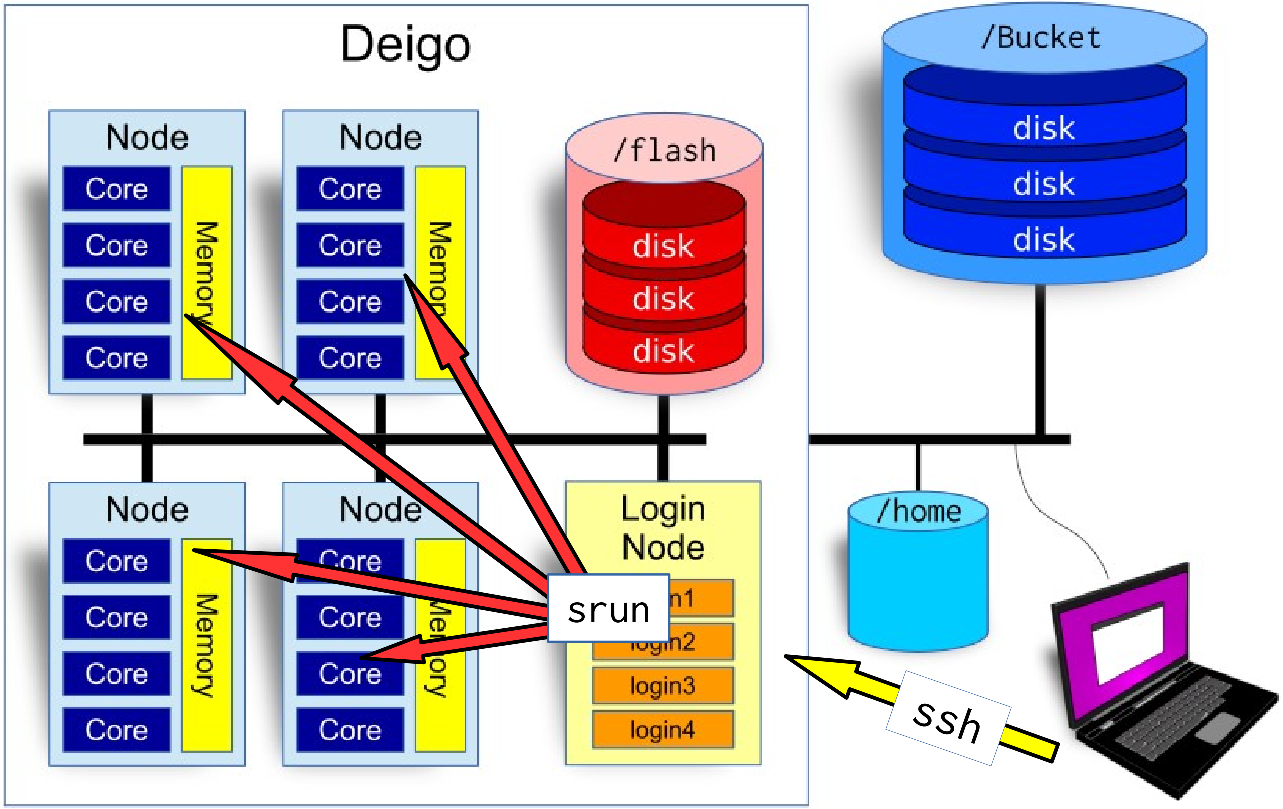


Press the tab key to fill in the name

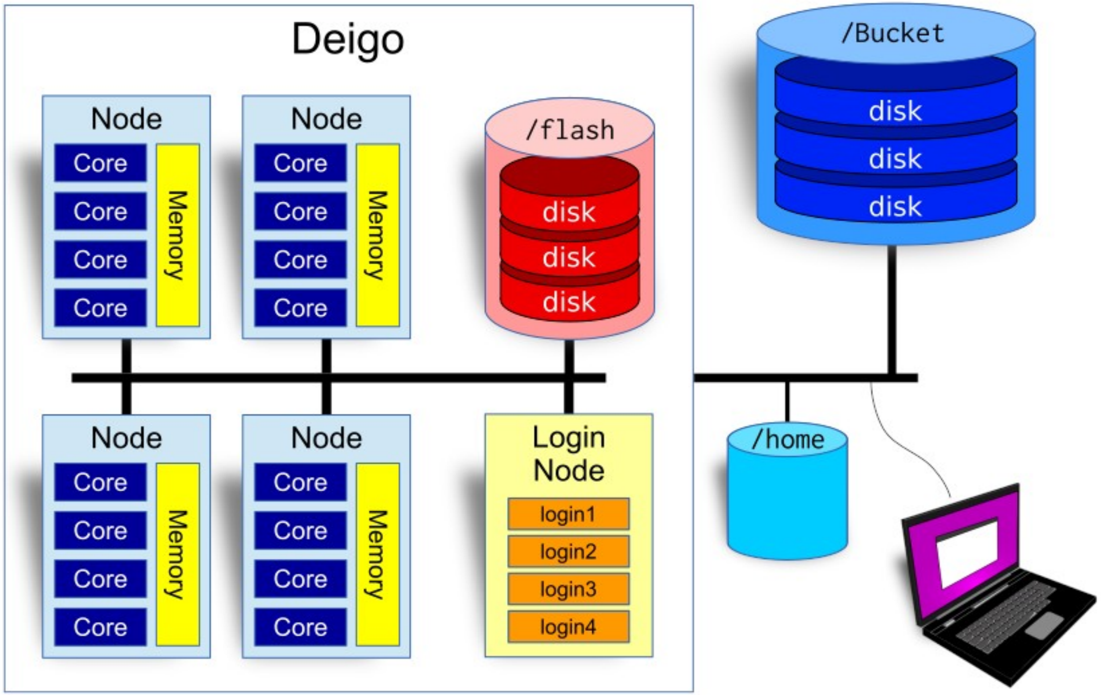
Press **once** to fill in unique parts.

Press **twice** to see matching alternatives. This works with directories, files, programs and parameters.

The Deigo cluster



Deigo Storage

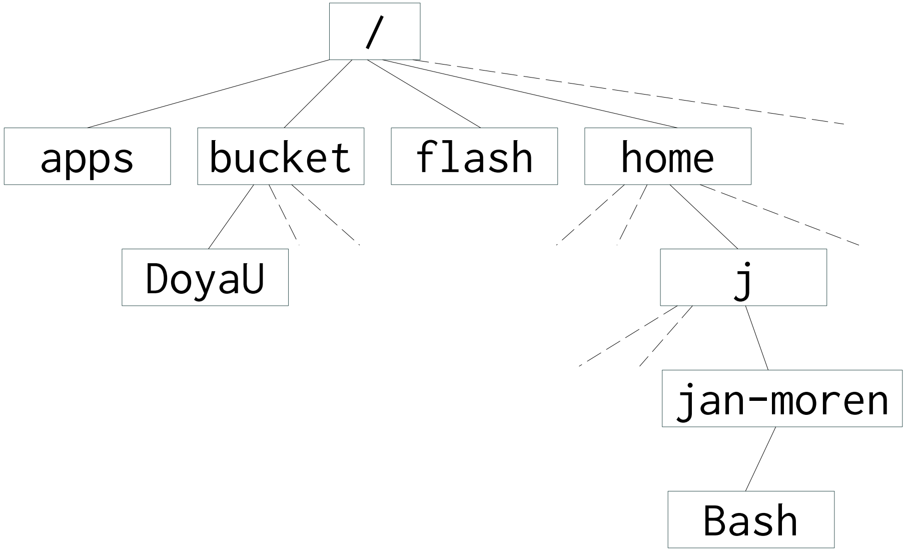


/home	Your home. Small (<50GB), slow. **Use for:** configuration files, source
/flash	In-cluster file system Not big (10TB/unit), fast **Use for:** running jobs
/bucket	Long-term storage file system Big, backed up read-only from computing nodes **Use for:** storing data
Comspace	
	5TB/unit Share storage across units, restricted storage etc.

The File System

Similar to OSX, different from Windows

- A single tree (no "c:", "d:")
- "/" is the folder separator
- "directory" = "folder"



The File System

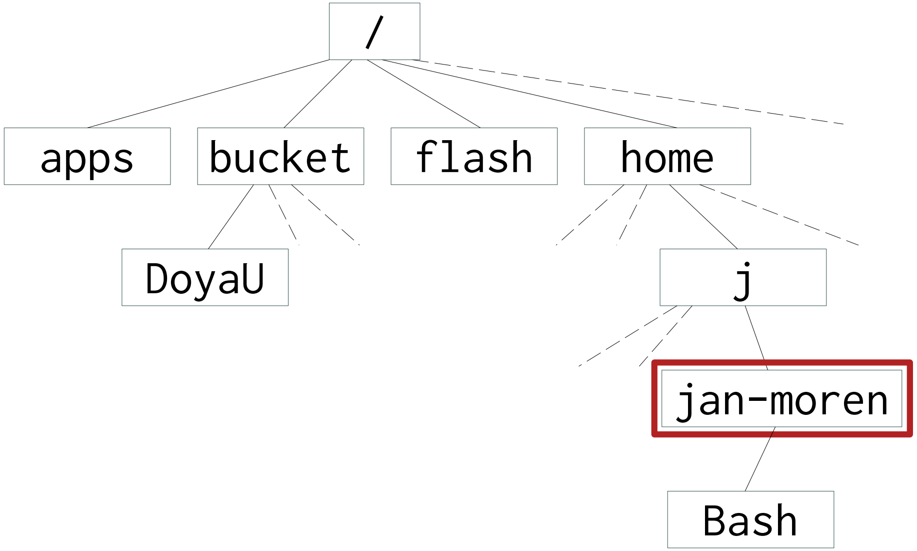
Similar to OSX, different from Windows

- A single tree (no "c:", "d:")
- "/" is the folder separator
- "directory" = "folder"

- You are always in a directory

  you start in your "home"
- See your current directory with "pwd":

```
$ pwd
/home/j/jan-moren
```



The File System

See your current directory with "pwd":

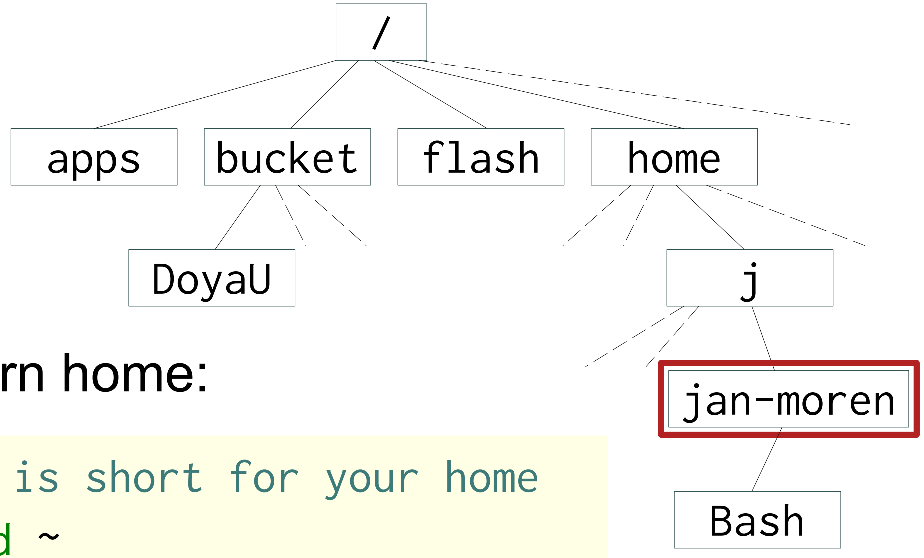
```
$ pwd
/home/j/jan-moren
```

List current directory with "ls":

```
$ ls
10.2                    nl3
```

Change directory with "cd":

```
$ cd Bash
```



Return home:

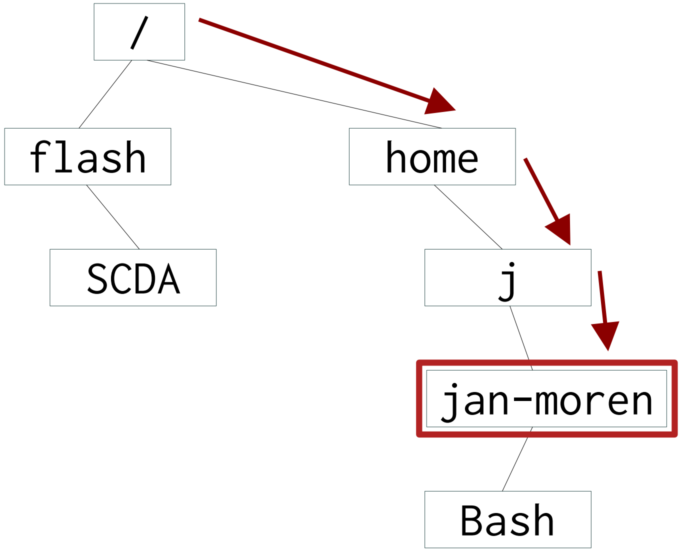
```
# ~ is short for your home
$ cd ~
# $HOME is also your home
$ cd $HOME
# just 'cd' returns you home
$ cd
```

The File System

- "/" is folder separator
- "directory" = "folder"
- You start in your "home"

Absolute path begins from the top with "/":

```
$ ls /home/j/jan-moren
```



The File System

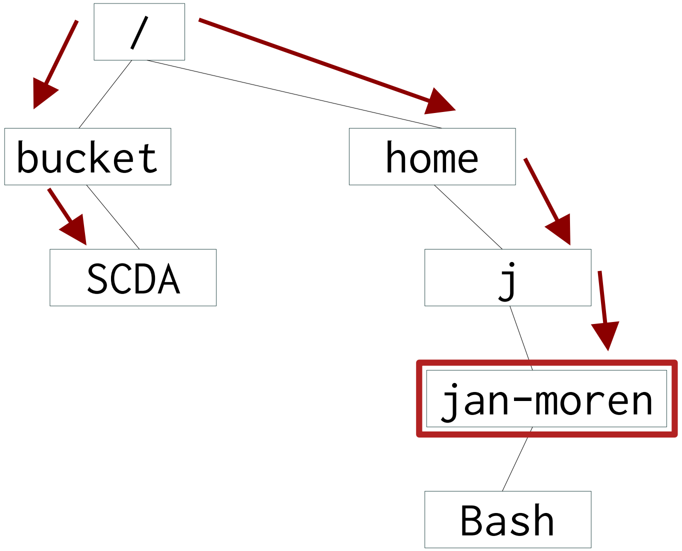
- "/" is folder separator
- "directory" = "folder"
- You start in your "home"

Absolute path begins from the top with "/":

```
$ ls /home/j/jan-moren
```

```
$ ls /bucket/SCDA
```

try with your unit

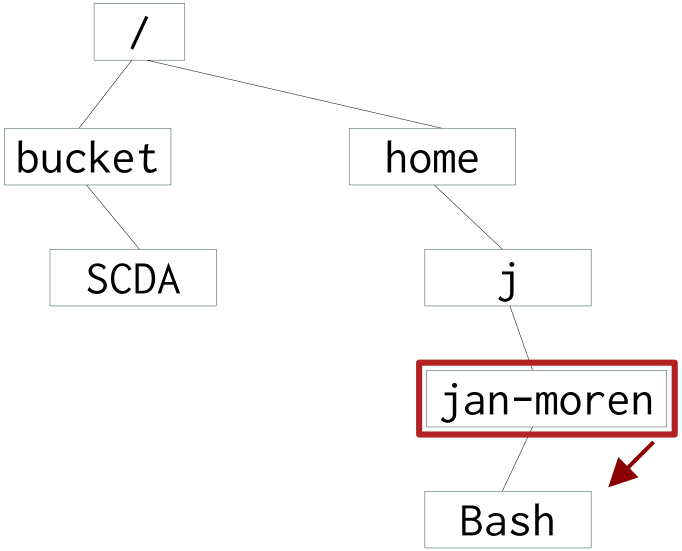


The File System

- "/" is folder separator
- "directory" = "folder"
- You start in your "home"

Relative path begins from where you are:

```
$ ls Bash
```



The File System

- "/" is folder separator
- "directory" = "folder"
- You start in your "home"

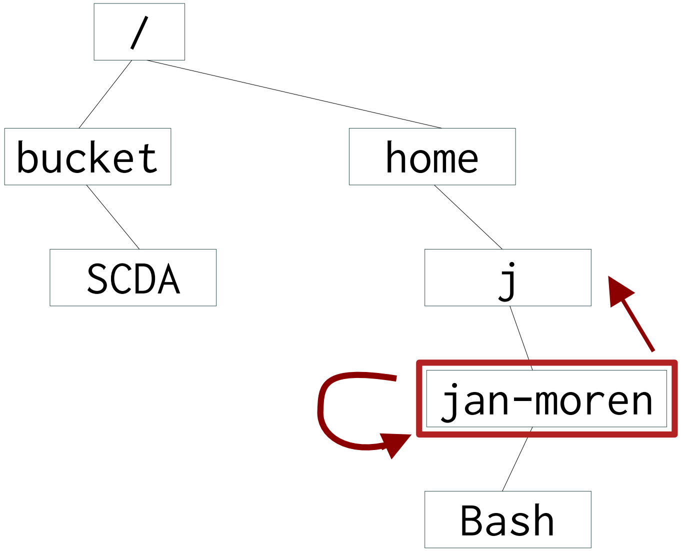
Relative path begins from where you are:

```
$ ls Bash
```

"." means here and ".." means one step up:

```
$ ls ..
```

```
# same thing:
$ ls
$ ls .
```



The File System

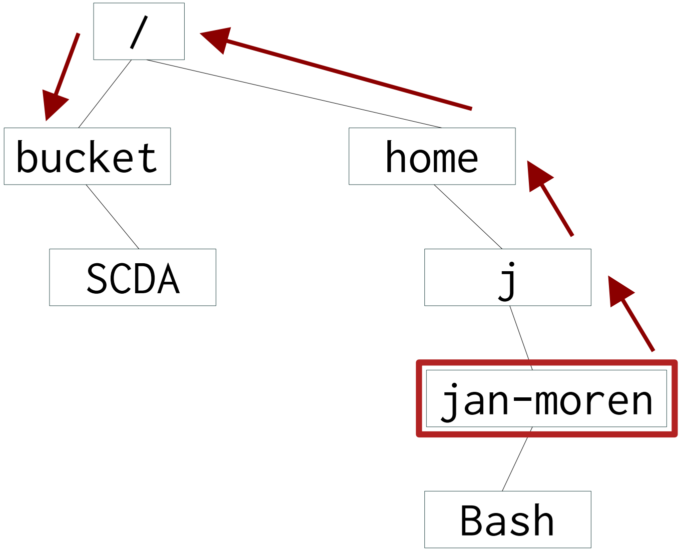
- "/" is folder separator
- "directory" = "folder"
- You start in your "home"

Relative path begins from where you are:

```
$ ls Bash
```

"." means here and ".." means one step up:

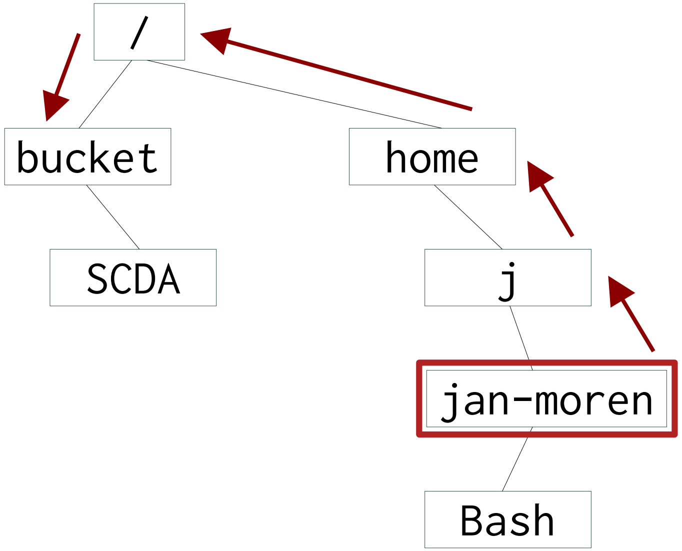
```
$ ls ../../../bucket
# same thing:
$ ls .././.././bucket
```



The File System

Useful Commands

`ls <path>`	list contents of directory
`cd <path>`	go to directory
`pwd`	show your current directory
`cp <source> <dest>`	copy file or directory
`mv <source> <dest>`	move (rename) file or directory
`rm <thing>`	delete file or directory
`mkdir <dir>`	create new directory
`touch <file>`	create new, empty file (or "update" a file)
`less <file>`	open and read a file
`nano <file>`	edit a file (limited, simple editor)



Wildcards

Match groups of files by name

- * stands for "any number of characters"
- ? stands for "any one character"

```
# any file ending with ".txt"
$ ls *.txt
macbeth.txt   bipp.txt      hello.txt
# a set of fasta files
$ ls name_ID????.fasta
name_ID0001.fasta     name_ID0002.fasta
```

Command line tips:

- **`history`** shows your command history
- ⬆️ and ⬇️ moves through your previous commands
- **`ctrl+r`** searches through your command history
- `!number` run command <number>

Help!!!!

1. Use the help options

Most (not all) commands have either
"-h" or "--help" for a short summary:

```
$ rm --help
Usage: rm [OPTION]... [FILE]...
Remove (unlink) the FILE(s).

  -f, --force    ignore nonexistent files and arguments, never prompt
  -i             prompt before every removal
  -I             prompt once before removing more than three files, or
                   when removing recursively; less intrusive than -i,
...
```

Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))

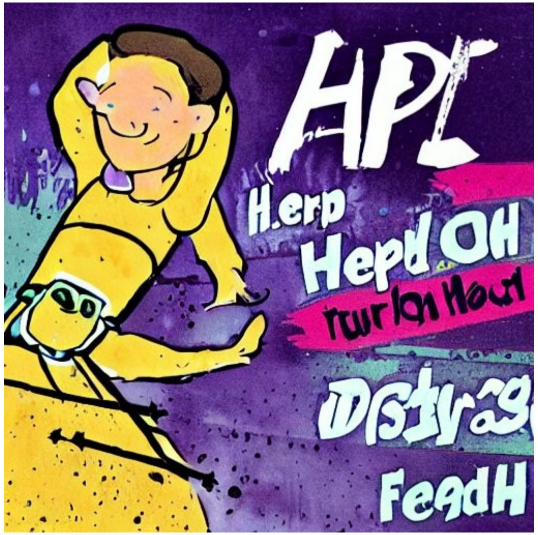
Help!!!!

2. "man" pages

many applications have manual pages with terse usage information in a standard format:

```
$ man rm
```

```
 dRM(1)                          User Commands                    RM(1)

NAME
    rm - remove files or directories

SYNOPSIS
    rm [OPTION]... [FILE]...

DESCRIPTION
    This  manual  page  documents the GNU version of rm. rm removes
    each specified file. By default, it does not remove directories.
. . .
```



man uses "less" to show text:

⬆️,⬇️	move up and down
<space>	page down
/	search for a string
q	quit

Help!!!!

3. Search online

- Many, many pages, tutorials, forums online.
- Google is still least bad (maybe)
- Use "bash" as one keyword

Do NOT just copy and paste commands you find online!

- It might be **outdated**
- It might be **wrong**
- It might be **malicious**

→ Always try to understand what you're doing

Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))

Exercise

It's Your Turn!

- Here is the cluster training data directory:

  `/apps/share/training/Intro`

- Under that directory, find a file called "`animals.txt`"

1. Create a new directory called "`from_intro`" inside your `Bash` directory in your home
2. Copy the file to your new directory
3. Find out how many Platypuses we have

```
$ man <command>
$ command --help
```

```
ls <path>
cd <path>
pwd
```

```
cp <source> <dest>
mv <source> <dest>
rm <thing>
```

```
mkdir <dir>
touch <file>
```

```
less <file>
nano <file>
```

Let’s set up Key-based login

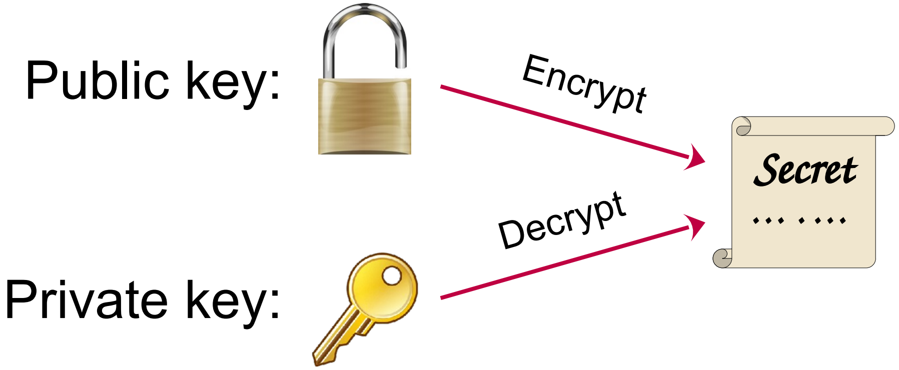
Passwords are a pain:

- Have to type every time you log in
  - … and every time you copy a file
- Not very secure
- Can’t log in from outside OIST

Keys are the solution!

- No need to remember anything
- No need to *type* anything
- Log in directly from anywhere

SSH Keys



Public key:

Encrypt

Decrypt

Private key:

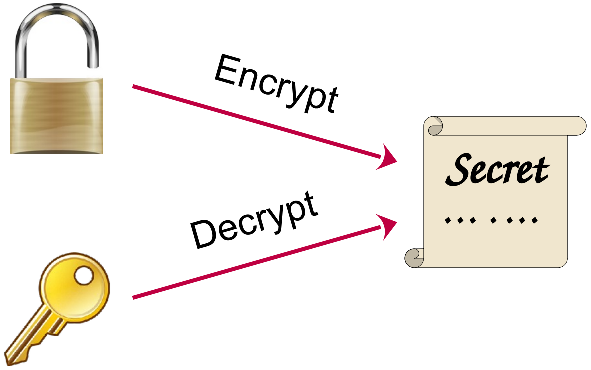
SSH keys use *asymmetric* encryption:

- One key can only *encrypt*
- The other can only *decrypt*
- *Anybody* can encrypt a message with the public key;
- *Only you* can decrypt it with the private key.

SSH Keys

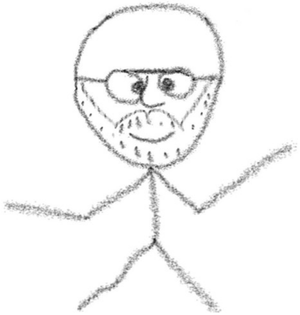
Public key:

Private key:

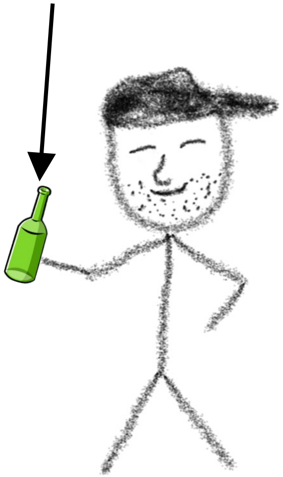


Encrypt

Decrypt



Me

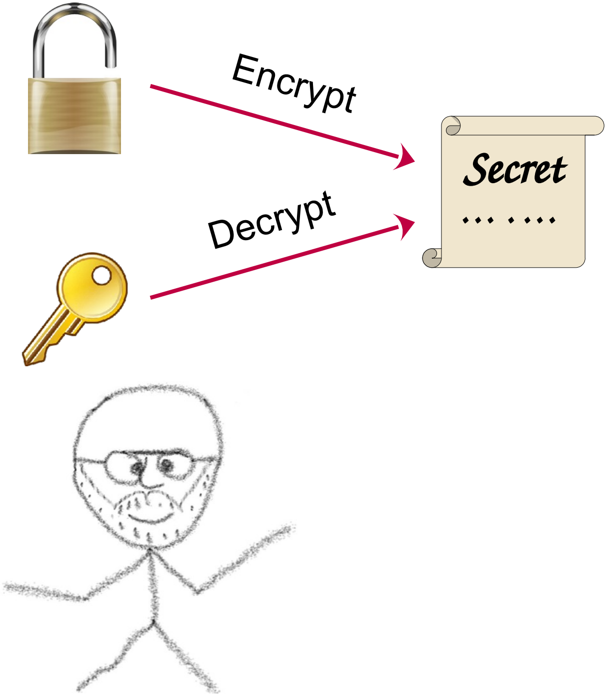


Apple cider

My brother

(artists impression)

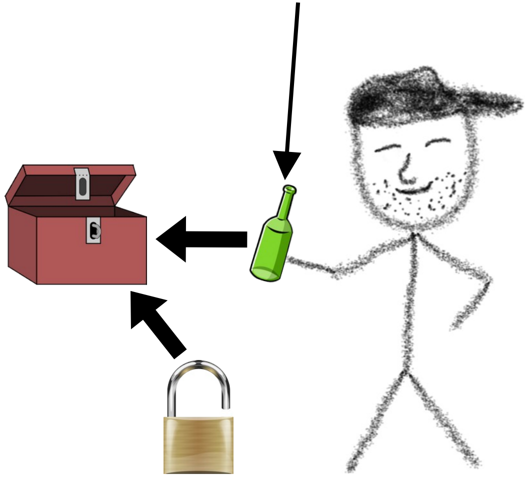
SSH Keys



Public key:

Private key:

Me

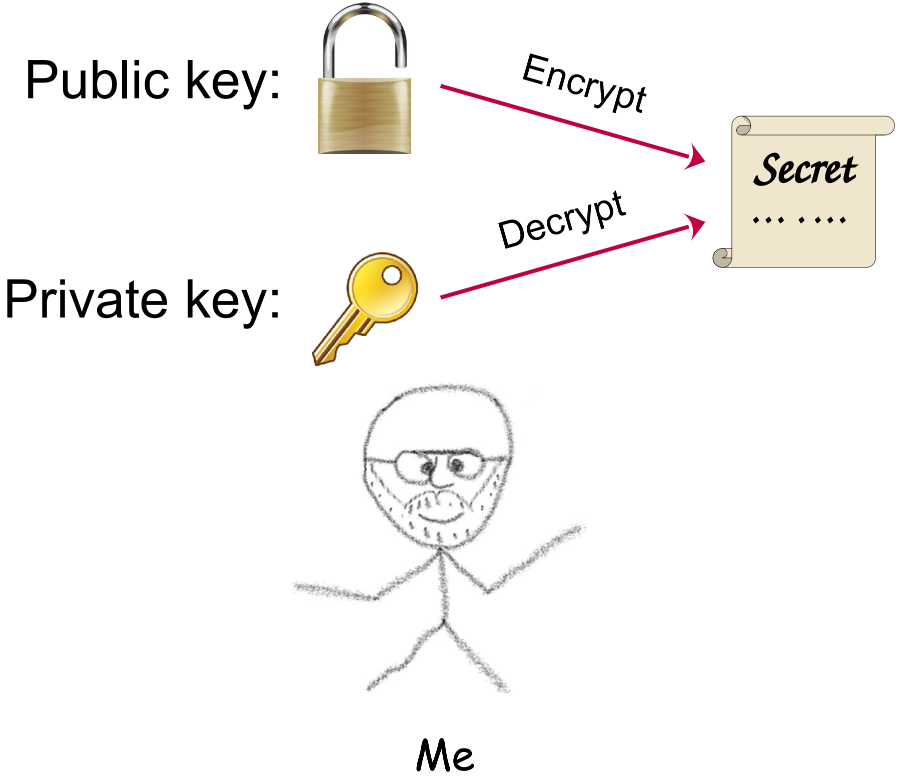


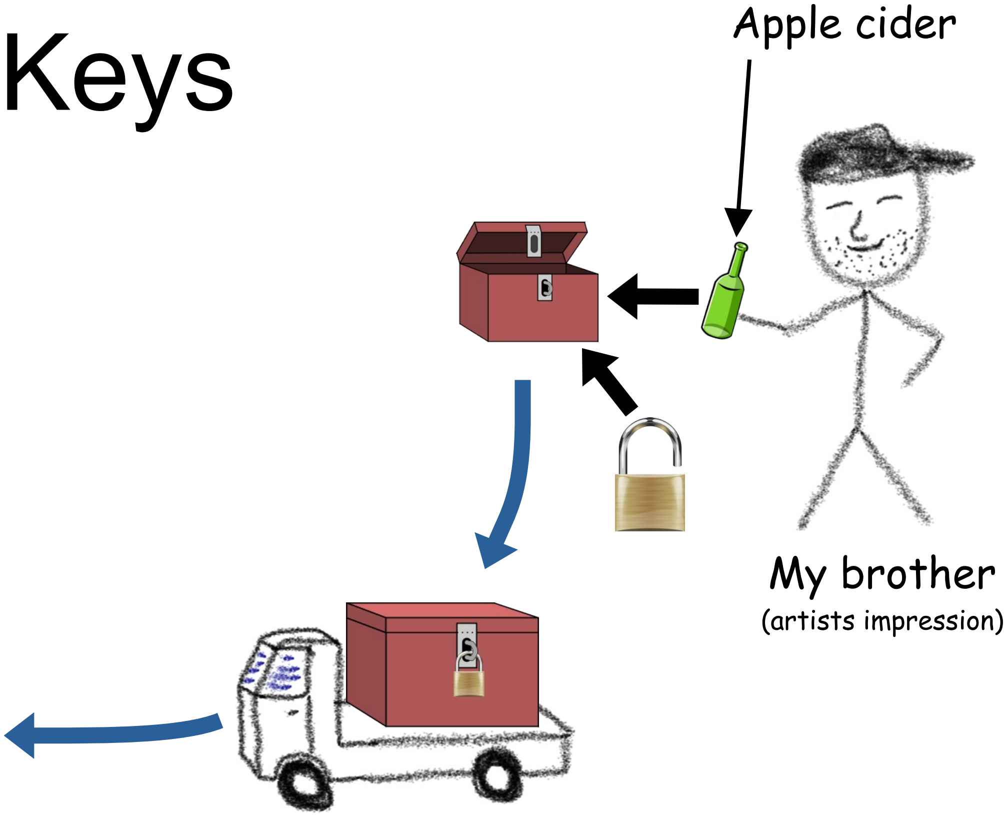
Apple cider

My brother

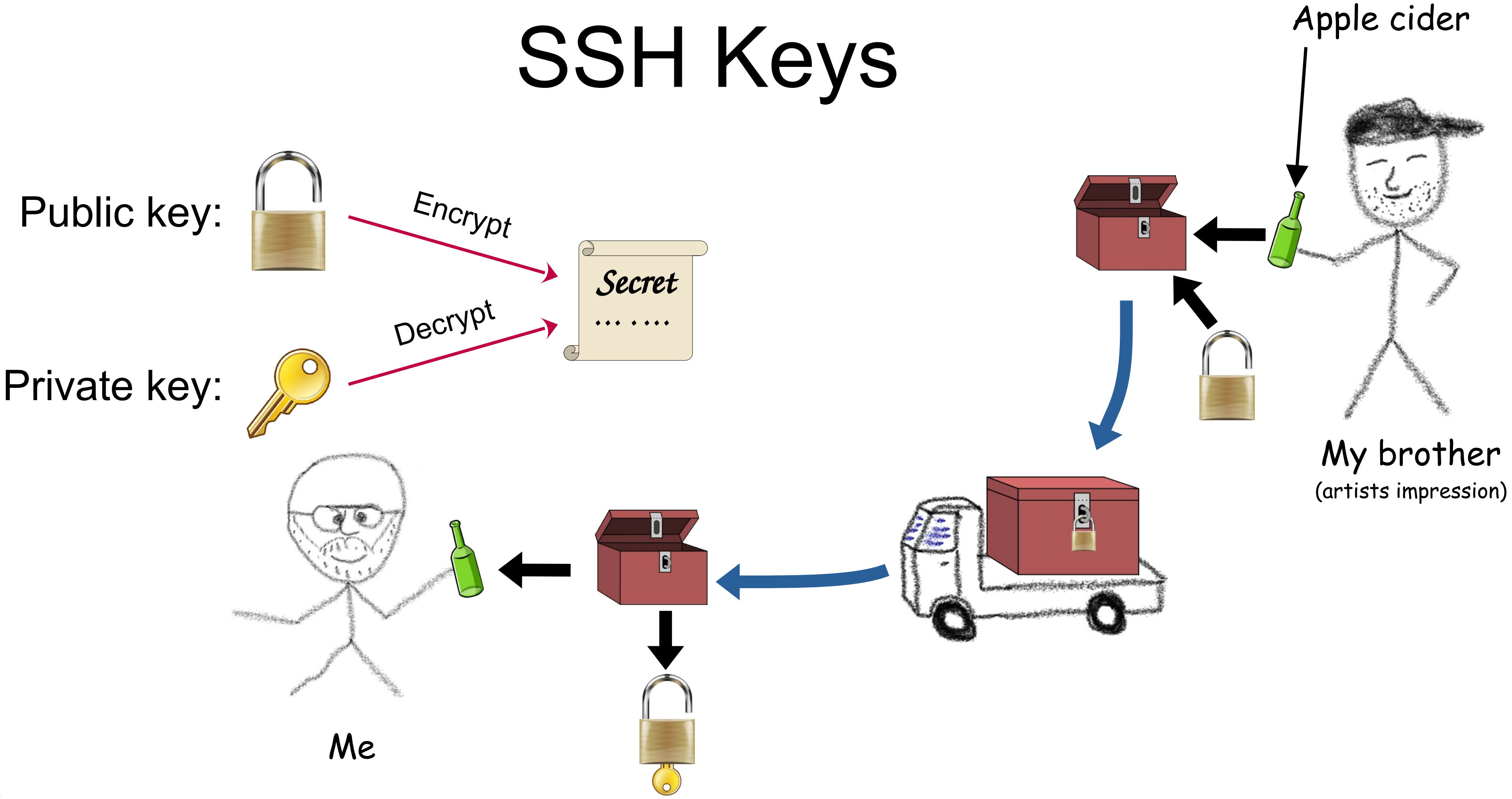
(artists impression)

SSH Keys


Public key:
Encrypt
Secret
... .
Decrypt
Private key:
Me


Apple cider
My brother
(artists impression)

SSH Keys



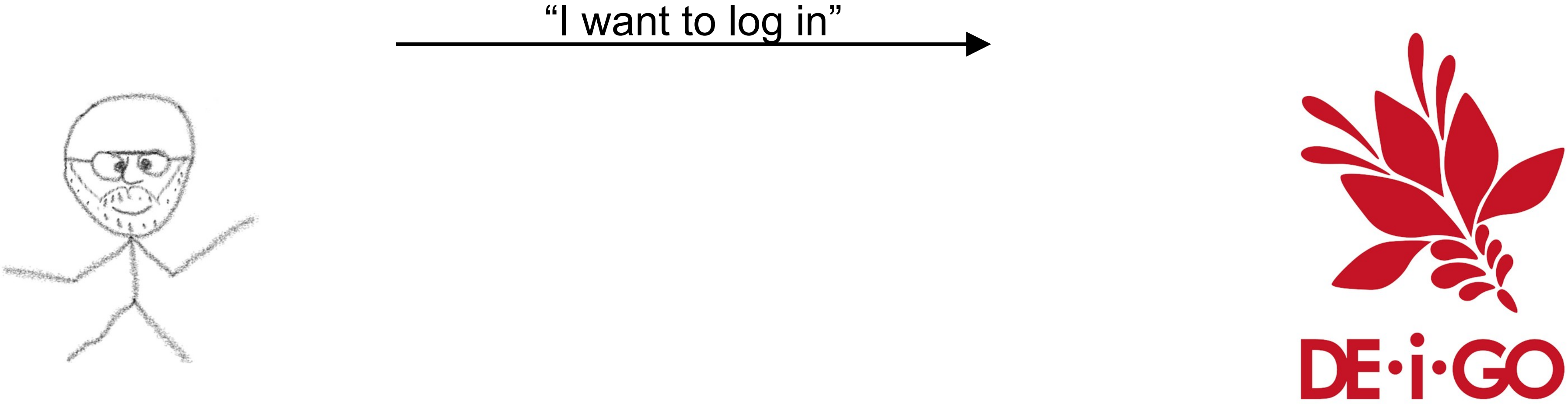
Public key:

Encrypt

Private key:

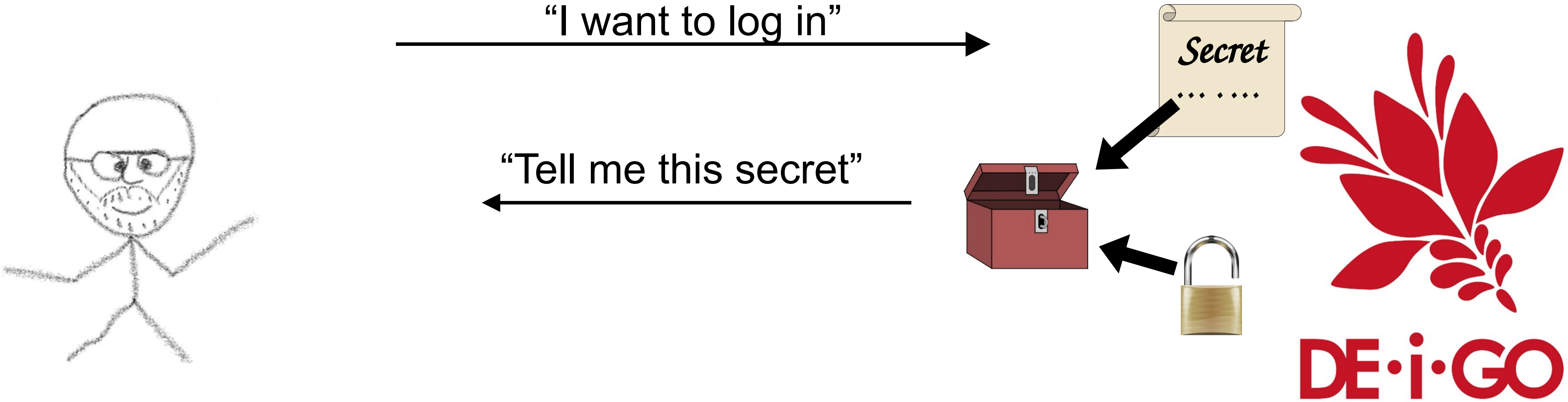
Decrypt

Prove Your Identity With Keys



“I want to log in”

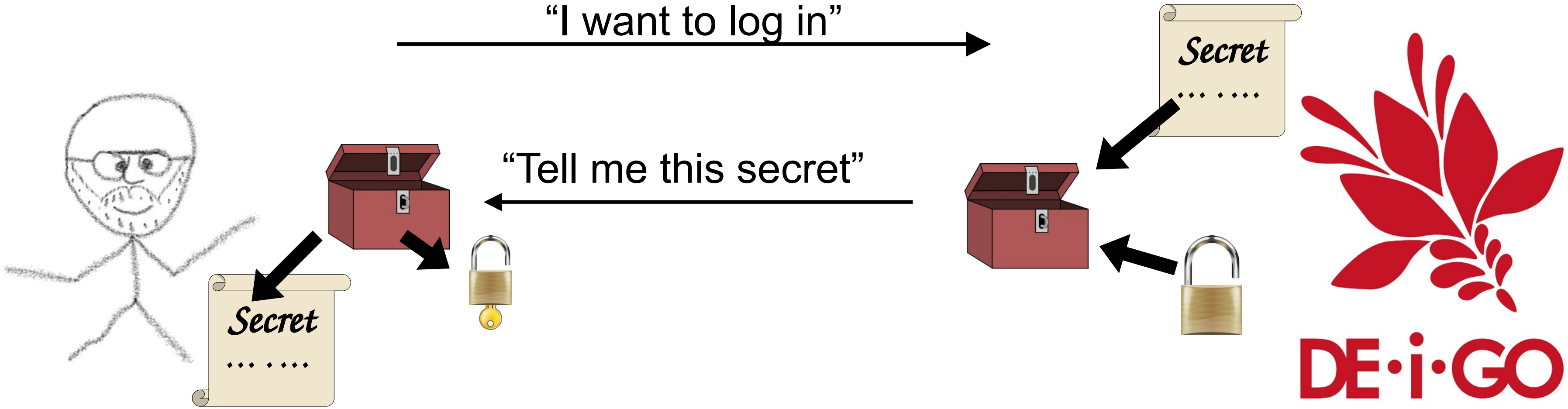
Prove Your Identity With Keys



“I want to log in”

“Tell me this secret”

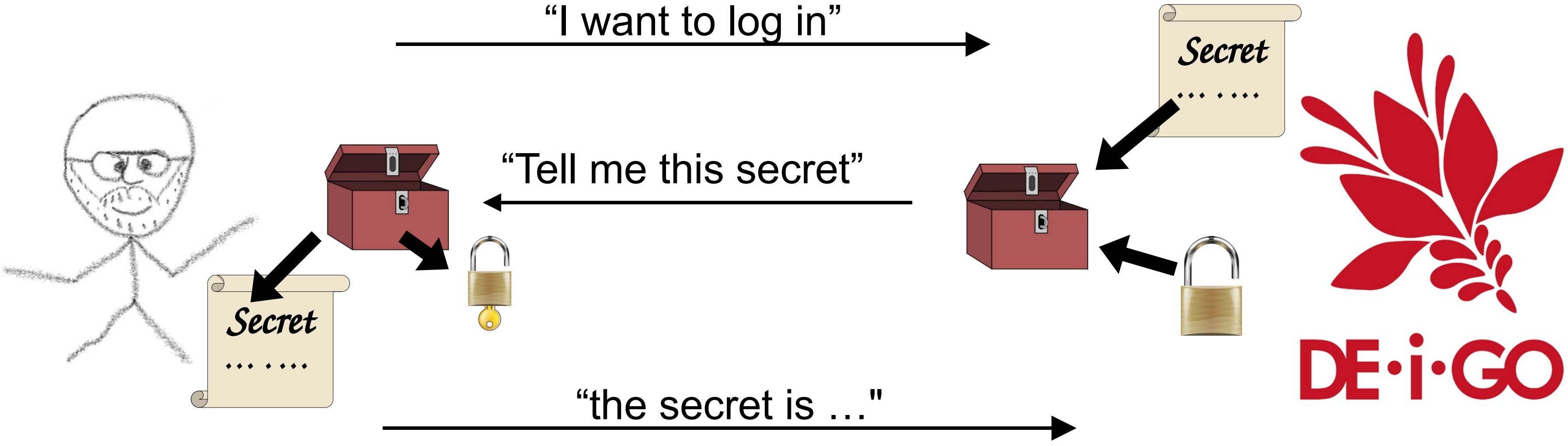
Prove Your Identity With Keys



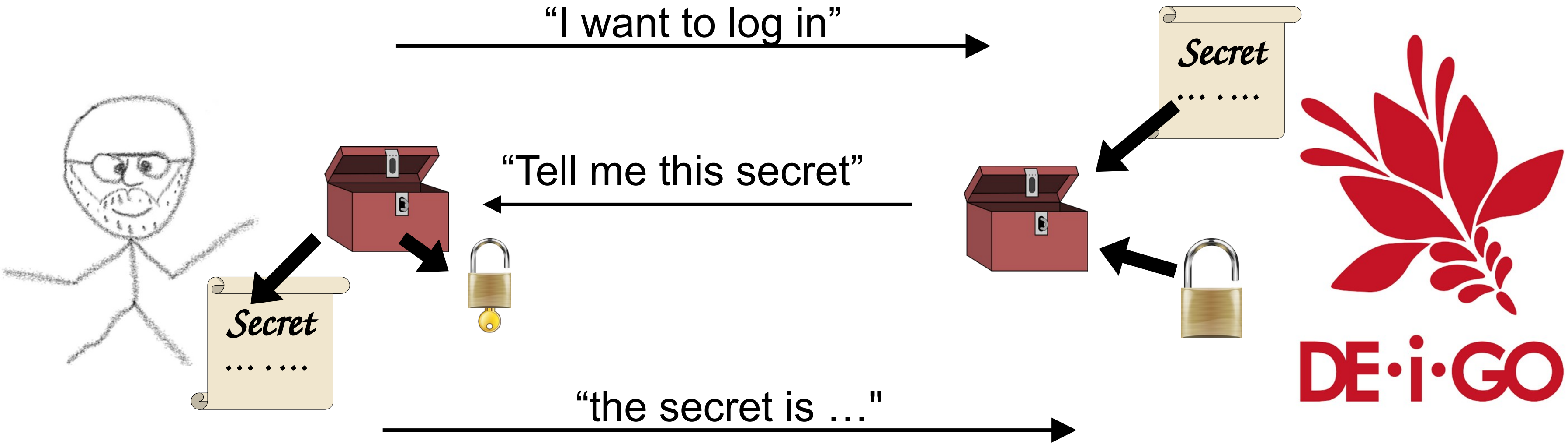
“I want to log in”

“Tell me this secret”

Prove Your Identity With Keys



Prove Your Identity With Keys



“I want to log in”

“Tell me this secret”

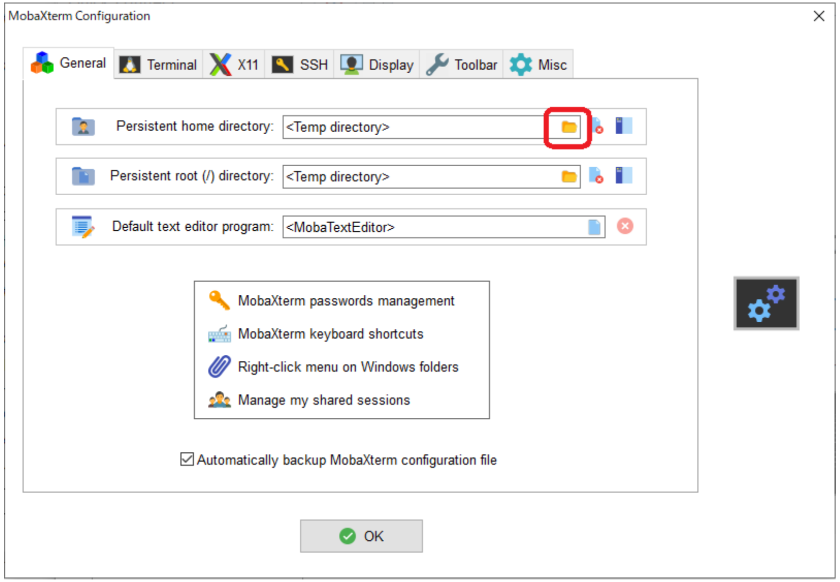
“the secret is …"

You *are* your private key

any machine with your private key can log
into any machine with your public key

Windows users

If you are using “Mobaxterm”, create a persistent home directory


MobaXterm Configuration

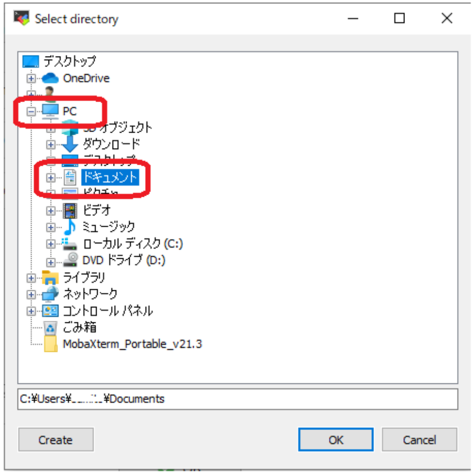
Persistent home directory: <Temp directory>

Persistent root (/) directory: <Temp directory>

Default text editor program: <MobaTextEditor>

☑ Automatically backup MobaXterm configuration file

General
Terminal
X11
SSH
Display
Toolbar
Misc
MobaXterm passwords management
MobaXterm keyboard shortcuts
Right-click menu on Windows folders
Manage my shared sessions
OK


Select directory

C:¥Users¥[redacted]¥Documents

Generate a Key Pair

Open a terminal ***on your laptop***, and run:

```
$ ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/xxxxx/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/xxxxx/.ssh/id_ed25519
Your public key has been saved in /home/xxxxx/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:XlMcATZdFfbK3cHNqACw9cpIdv89LUgzHvUBMYqIJYo janne@loke
...
```

Press “enter”

Copy The Key To Deigo

on your laptop, run:

```
$ ssh-copy-id user-name@deigo.oist.jp
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s),
to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed --
if you are prompted now it is to install the new keys
user-name@deigo.oist.jp's password:
...
```

Test the New Key

on your laptop, run:

```
$ ssh user-name@deigo.oist.jp
 ******************************************************************************
 *  Unauthorized access to this resource is prohibited.                       *
 *  Okinawa Institute of Science and Technology.                              *
 ******************************************************************************
Last login: Wed Aug  7 14:37:36 2024 from 10.13.69.218
jan-moren@deigo-login4: (10:33)
~$
```

Note: you *must* be on the “OIST” network!

Create an SSH Config File

Many apps can be configured with configuration files

- They’re usually text format
- They are usually in your home:
  - A file beginning with a ‘.’
  - In an app-specific subfolder
  - In the ‘.config/’ subfolder

Names that begin with ‘.’ are hidden by default

```
# the -a option shows hidden files/dirs
$ ls -a
.bash_history           # your shell history
.bash_profile           # Bash settings
.bashrc                 # Bash settings
.cache/                 # temporary data
.config/                # many apps settings
.inputrc                # commandline settings
.lmod.d/                # Your module cache
.matlab/                # matlab settings
.ssh/                   # Your SSH folder
.vimrc                  # Vim editor settings
```

Create an SSH Config File

We need an editor......a ***code*** editor. What is that?

Uses *only* plain text

Text editors change simple “ to fancy “ and ”, plain – (minus) to fancy — and so on.

These are completely different characters, so your code breaks.

Supports syntax highlighting, autocompletion and so on

Text editors (avoid these):

- Word
- Wordpad
- TextEdit

Code editors:

- Notepad
- VSCode
- SublimeText

Create an SSH Config File

Lots of options on the cluster!

- **nano** - very simple, easy but limited
- **gedit** - modern, simple, mouse support but needs graphics support
- **vim** - classic, very powerful, available everywhere but difficult to learn
- **VSCode** - popular, run locally and edit on the cluster, a bit complex

Nano is already installed on **MacOS** and on **Linux**.

Install in **MobaXterm** with:

```
$ apt install nano
```

Nano benefits:

- Available everywhere
- Text mode – no graphics needed
- Runs fine across SSH
- Simple and easy to grasp

Create an SSH Config File

on your laptop, run:

```
$ nano .ssh/config
```

- Add the text to the right

  You can copy and paste from our documentation site

- Replace “your-id” with your OIST ID
- Replace ‘id_ed25519’ with your key name if needed.

```
User your-id
IdentityFile ~/.ssh/id_ed25519
ForwardX11 yes


Host deigo
    Hostname deigo.oist.jp
Host saion
    Hostname saion.oist.jp
host oist-ext
    hostname login.oist.jp
host deigo-ext
    ProxyCommand ssh -q -W deigo.oist.jp:22 oist-ext
host saion-ext
    ProxyCommand ssh -q -W saion.oist.jp:22 oist-ext
```

Test your Config!

First, run this:

```
$ ssh deigo

*****************************************
*
*  Unauthorized access to this resource is
*  Okinawa Institute of Science and Techno
*
*****************************************

Last login: Tue Aug 20 11:08:24 2024 from
jan-moren@deigo-login3 $
```

If it worked, try to access Deigo from outside OIST:

Test access from outside:

1. Connect to "**OIST-Public**"
2. Then run:

```
$ ssh deigo-ext

*****************************************
*
*  Unauthorized access to this resource is
*  Okinawa Institute of Science and Techno
*
*****************************************

Last login: Tue Aug 20 11:08:24 2024 from
jan-moren@deigo-login3 $
```

Create an SSH Config File

These are set globally

```
User your-id
IdentityFile ~/.ssh/id_ed25519
ForwardX11 yes

Host deigo
    Hostname deigo.oist.jp
Host saion
    Hostname saion.oist.jp
host oist-ext
    hostname login.oist.jp
host deigo-ext
    ProxyCommand ssh -q -W deigo.oist.jp:22 oist-ext
host saion-ext
    ProxyCommand ssh -q -W saion.oist.jp:22 oist-ext
```

If you want different settings for different hosts, move them into each hosts settings:

```
Host deigo
    User your-id
    Hostname deigo.oist.jp
    IdentityFile ~/.ssh/id_ed25519
    ForwardX11 yes
Host mycomputer
    User your-other-id
    Hostname mycomputer.com
    IdentityFile ~/.ssh/my_other_key
```

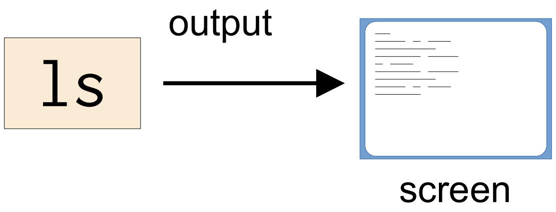
Redirection and Pipes

You want to read the too-long output of a command:

```
# List all programs in /usr/bin
$ ls /usr/bin
```

Way too long…

Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))



ls

output

screen

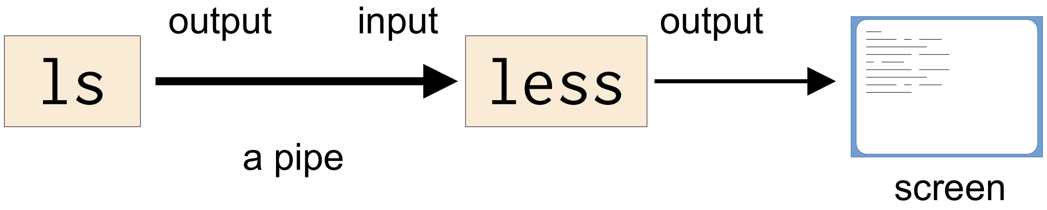
Redirection and Pipes

You want to read the too-long output of a command:

```
# List all programs in /usr/bin
$ ls /usr/bin
```

Way too long… We want to use "`less`" to view it.
We can do that with a *pipe*:

```
# List all programs in /usr/bin
$ ls /usr/bin | less
```



Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))

Redirection and Pipes

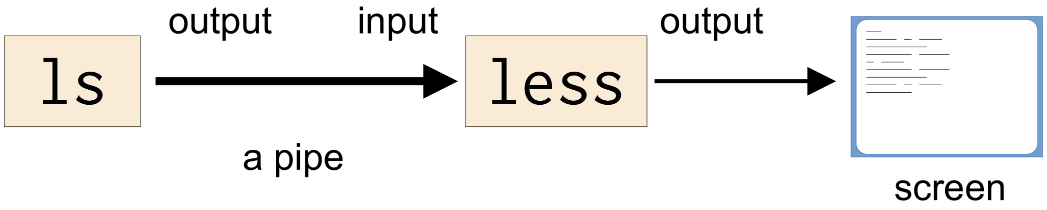
You want to read the too-long output of a command:

```
# List all programs in /usr/bin
$ ls /usr/bin
```

Way too long... We want to use "`less`" to view it.
We can do that with a *pipe*:

```
# List all programs in /usr/bin
$ ls /usr/bin | less
```

Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))



input	= `stdin`
output	= `stdout`
error	= `stderr`

Redirection and Pipes

You can redirect to and from a file with '<' and '>':

```
# List all programs in /usr/bin, redirect to file
$ ls /usr/bin >binfiles.txt

# Redirect the file to less
$ less <binfiles.txt
```

Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))

input = `stdin`
output = `stdout`
error = `stderr`

Redirection and Pipes

You can redirect to and from a file with '<' and '>':

```
# List all programs in /usr/bin, redirect to file
$ ls /usr/bin >binfiles.txt

# Redirect the file to less
$ less <binfiles.txt
```

You can add to a file with '>>':

```
# add a monthly report to the year
$ cat September.txt >>year2023.txt
```

Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))

input = `stdin`
output = `stdout`
error = `stderr`

Redirection and Pipes

You can redirect to and from a file with '<' and '>':

```
# List all programs in /usr/bin, redirect to file
$ ls /usr/bin >binfiles.txt

# Redirect the file to less
$ less <binfiles.txt
```

You can add to a file with '>>':

```
# add a monthly report to the year
$ cat September.txt >>year2023.txt
```

redirect both stdout and stderr with '&>':

```
# build some program, save the output for later:
$ make &>build.log
```

Credit: Stable Diffusion
(and a few hours of procrastination
(I really meant to be working but I got distracted by the shiny lights))

input = stdin
output = stdout
error = stderr

Variables

- Variables contain values, typically text
- Set them like this:

```
# set a variable - no spaces around the ”=”
$ value=”42”
```

- Get the value with `${value}` or `$value`
- You can print stuff using "`echo`"

```
# get a variable value
$ echo ${value}
42

# same but simpler to type (but a bit ambiguous)
$ echo $value
42
```

Credit: Bing Create

(They really suck at generating image of code, obviously)

Variables

- First ${…} expands into the *content* of the variable
- Then the line is evaluated

```
# let’s create a file name:
$ value=”42”

# when run, ${value} is first replaced with 42:
$ echo gene${value}a_01.fasta
```

Credit: Bing Create

(They really suck at generating image of code, obviously)

Variables

- First ${…} expands into the *content* of the variable
- Then the line is evaluated

```
# let’s create a file name:
$ value=”42”

# when run, ${value} is first replaced with 42:
$ echo gene42a_01.fasta
```

Credit: Bing Create

(They really suck at generating image of code, obviously)

Variables

- First ${…} expands into the *content* of the variable
- Then the line is evaluated

```
# let’s create a file name:
$ value="42"

# when run, ${value} is first replaced with 42:
$ echo gene42a_01.fasta
gene42a_01.fasta
```

Credit: Bing Create

(They really suck at generating image of code, obviously)

OIST

Variables

- First ${…} expands into the *content* of the variable
- Then the line is evaluated

```
# let's create a file name:
$ value="42"

# when run, ${value} is first replaced with 42:
$ echo gene42a_01.fasta
gene42a_01.fasta
```

- If we used just $value here:

```
# Now bash thinks the variable is valuea_01
$ echo gene$valuea_01.fasta
gene.fasta
```

Credit: Bing Create

(They really suck at generating image of code, obviously)

Environment Variables

- Ordinary variables only visible to the script itself
- *Environment variables* visible to all child programs
- Used for various general settings

`HOME`	Your home directory
`USER`	Your user name
`LANG`	Your language
`PATH`	List of directories to look for programs
`HOSTNAME`	Name of the current computer / node

```
# make it an environment variable
$ export value
# see environment
$ env | less
```

Credit: Bing Create

(They really suck at generating image of code, obviously)

Environment Variables

- Ordinary variables only visible to the script itself
- *Environment variables* visible to all child programs
- Used for various general settings

HOME	Your home directory
USER	Your user name
LANG	Your language
PATH	List of directories to look for programs
HOSTNAME	Name of the current computer / node

```
# Add a path to PATH:
$ PATH="/new/path/name:$PATH"
# export new PATH into environment
$ export PATH
```

Credit: Bing Create

(They really suck at generating image of code, obviously)

.bashrc

The main shell configuration file

- Set parameters
- create aliases
- set environment variables
- Change your prompt
- …

File names starting with "." are normally hidden.

"`ls -a`" ("all") will show them

```
# Use "nano" to edit .bashrc
$ nano .bashrc

# use 'source' to re-read the file
$ source .bashrc
```

```
# example settings
# correct minor directory errors
shopt -s cdspell
# set history length
HISTSIZE=15000
HISTFILESIZE=15000

# alias example
alias ll="ls -trl"         # long form ls
```

Questions?
Anything you want to know?

Find our "introduction to Bash" page in our documentation:

https://groups.oist.jp/scs
/command-line-introduction-bash

Our "advanced Bash" page cover lots of useful tools and patterns:

https://groups.oist.jp/scs
/advanced-bash

See the newest files:

```
# long list, with newest file at the end
$ ls -trl
# define as an alias in .bashrc
alias ll='ls -trl'
```

What's taking all the space?

```
# Size of all folders, in human readable
# format, sorted with largest at bottom
$ du -ch|sort -h
```

Questions?
Anything you want to know?

Find our "introduction to Bash" page in our documentation:

https://groups.oist.jp/scs/command-line-introduction-bash

Our "advanced Bash" page cover lots of useful tools and patterns:

https://groups.oist.jp/scs/advanced-bash

Find strings with "grep"

```
# find Platypus in our animal list
$ grep ”Platypus” animals.txt
# Commands that begin with b in /usr/bin
$ ls /usr/bin | grep ”^b”
```

...also:

```
# count lines with ”Macbeth” in macbeth.txt
$ grep ”macbeth” macbeth.txt | wc -l
# date prints time, time measures time taken
$ time sleep 5
```

Editors

Lots of options!

- **nano** - very simple, easy but limited
- **gedit** - modern, simple, mouse support but needs graphics
- **vim** - classic, very powerful, available everywhere but difficult to learn
- **Vscode** - popular, run locally and edit on the cluster, a bit complex

If you edit locally on Windows, there is a problem:

Windows uses different end of line characters than Linux and Mac

Solution:

- either set your Windows editor to use "unix" or "LF" endings; or
- convert on cluster using "tr":

```
$ tr -d '\r' <win_file >linux_file
```